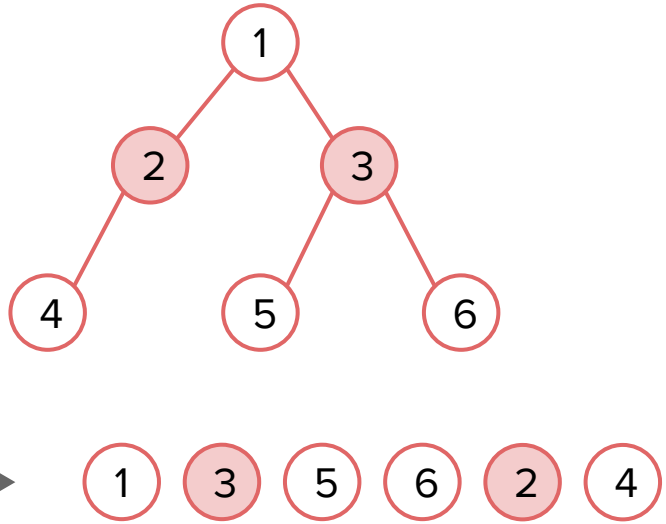
I: Double 01 on Tree

原案 : riantkb
問題文 : riantkb
データセット : hint908
解答 : hint908, hos, riantkb
解説 : riantkb

問題概要

- 01 on Tree という問題がある



01 on Tree の概要

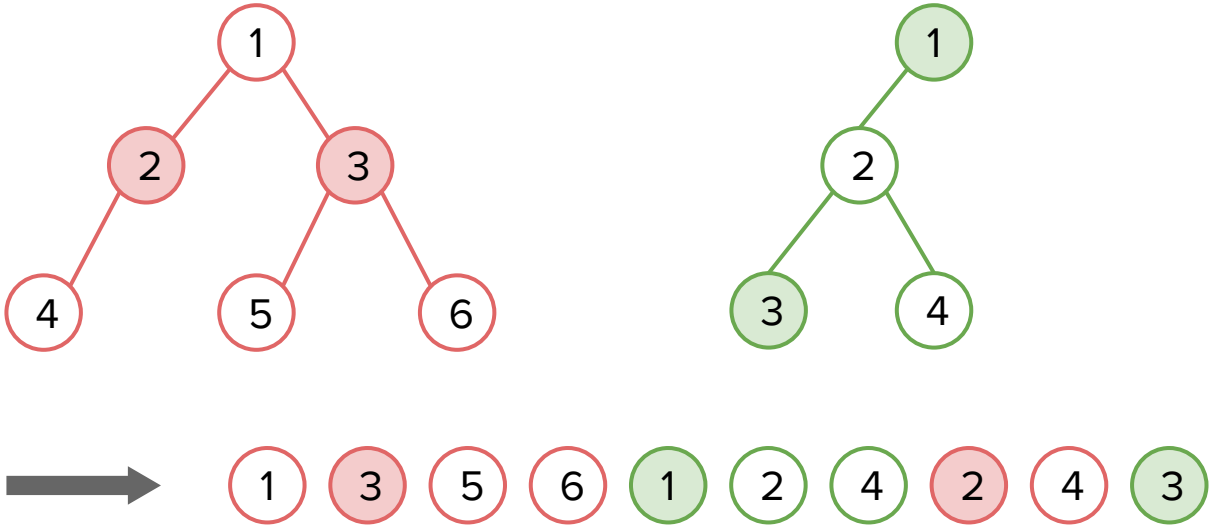
N 頂点の根付き木および 01 列 $(V_1, V_2, \ldots, V_N)$ が与えられる。
次の条件を満たす順列 $(q_1, \ldots, q_N)$ であって $(V_{q_1}, V_{q_2}, \ldots, V_{q_N})$ の転倒数を最小化するものを求めよ。

- $i = 2, \ldots, N$ について、i の親頂点の番号が i よりも q において左に登場する。

https://atcoder.jp/contests/abc376/editorial/11196 より引用

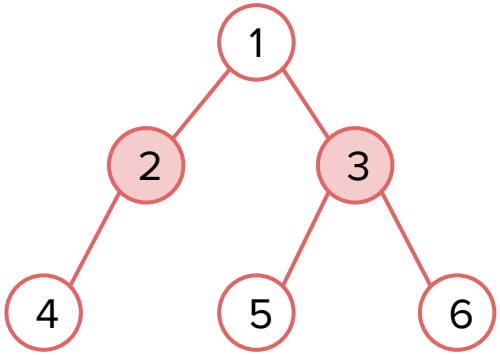
問題概要

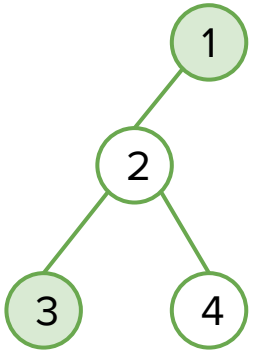
- これを、2 つの木に対して行う
  - これ自体は、超頂点をひとつ作り、そこからそれぞれの木を生やすのと同じ

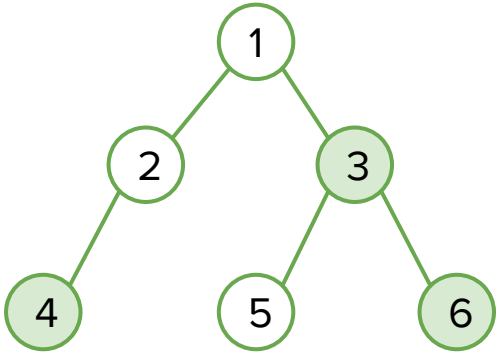


問題概要

- ただし、そのうち片方については Q 種類ある
  - しかもオンライン





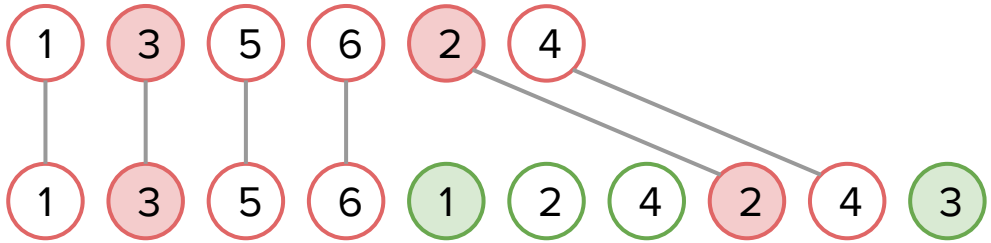


問題概要(制約)

- 1 <= N <= 2 * 10^5
  - N: 赤い木の頂点数
- 1 <= Q <= 10^5
  - Q: 緑の木の数
- sum(M_k) <= 2 * 10^5
  - sum(M_k): 緑の木の頂点数の総和

観察

- Double 01 on Tree 内での片方の木の頂点の並べる順序は、その木単体での 01 on Tree での並べる順序が保存される
  - その木の中での最適でない順序を採用していた場合、最適な順序にすることで必ず改善されるため
- どんな位置にもう片方の頂点が挿入されるのだろう？



01 on Tree への理解

- 01 on Tree で行う「2 頂点のマージ」は何をしているのか？

- ➡ この（マージされた）頂点の直後に
  この（マージされた）頂点を並べて損しない、ということを言っている

- ただし、Double 01 on Tree において単純に片方の木をすべて並べた後
  もう片方の木をすべて並べる、とすると当然ダメ

- これはどうして損をしている？

01 on Tree への理解

- 01 on Tree での「2 頂点のマージ」には以下の 2 通りがある

- par[v] >= v
  - 自分が親より順序が先（先に並べたい）ケース
- par[v] < v
  - 自分が親より順序が後（後に並べたい）ケース
  - **このケースは、par[v] が根のときにしか発生しない**
    - そうでない場合、par[v] とその親とのマージを先にするので

    - Double 01 on Tree ではさらに超頂点が存在するので、
      これが発生すると最適でなくなることがある

01 on Tree への理解

- par[v] >= v なマージだけを行うとどうなる？
  - お気持ち：これを並べるならこれを並べても絶対に損しない
  - 例：ある頂点を並べたなら、そこから直接行ける 0 は貪欲に並べてよい

- すべての v について、par[v] < v を満たすような木が得られる

- これに対する 01 on Tree の解法はどうなる？
  - 木の構造を無視してひたすら貪欲に取る、で問題ない
  - つまり、木ではなくソートされた列として解釈してよい

解法

- 先ほどまでの議論で、木をソート列に変換することができた
- つまり、Double 01 on Tree は以下のような問題に帰着できる
- ソート列が 2 つあるので、マージソートしてください
  - ただし、列のうち片方はたくさん与えられる
- これは、クエリの要素それぞれについてソート列で二分探索して
  挿入されるべき場所を見つければよい

実装方針に関して

- 01 on Tree の実装としてよく知られている方法でも解ける（はず）
  - よく知られている方法とは？
    - 全体で最小の頂点をその親とマージする、union-find を使う方法
  - priority_queue から出したときにマージしてよいかを判定すればよい
- priority_queue を持ちながら bottom up にマージする方法がおすすめ
  - この実装だと、最初からソート列の形で結果が得られる
  - BOJ で 01 on Tree like な問題を埋めていたときに見た koosaga の実装がこんな感じで
    それで知りました

実装方針に関して

```cpp
struct S {
    int c0, c1;
    S() : c0(0), c1(0) {}
    S(int c0, int c1) : c0(c0), c1(c1) {}

    friend bool operator<(const S& l, const S& r) {
        return l.c1 * (long long)r.c0 < l.c0 * (long long)r.c1;
    }
    friend bool operator>(const S& l, const S& r) {
        return l.c1 * (long long)r.c0 > l.c0 * (long long)r.c1;
    }
    static S merge(const S& l, const S& r) {
        return S(l.c0 + r.c0, l.c1 + r.c1);
    }
};

using p_que = priority_queue<S, vector<S>, greater<S>>;

p_que dfs(int p, const vector<S>& s, const vector<vector<int>>& children, long long& sum) {
    p_que pq;
    for (auto&& e : children[p]) {
        auto ch = dfs(e, s, children, sum);
        if (pq.size() < ch.size()) {
            swap(pq, ch);
        }
        while (!ch.empty()) {
            pq.push(ch.top());
            ch.pop();
        }
    }
    auto ps = s[p];
    while (!pq.empty() && !(ps < pq.top())) {
        sum += ps.c1 * (long long)pq.top().c0;
        ps = S::merge(ps, pq.top());
        pq.pop();
    }
    pq.push(ps);
    return pq;
}
```

ジャッジ解

- hint (C++) 135 lines, 3.1 kB
- hos (C++) 346 lines, 9.7 kB
- riantkb (C++) 118 lines, 2.8 kB

統計情報

- AC teams / Trying teams
  - 0 + ? / 4
- First Acceptance
  - ??? (??? min)